Computing Lab #4: Simple File I/O and arrays

Due: Tuesday, October 17, 2017 (midnight)

Programming language: Write, compile, and run a Fortran, C or C++ program. Your choice! This lab assumes the C programming.

Points: 10

I. TASK #1: ARE MY RANDOM POINTS ANY GOOD? (5 POINTS)

Task: Write, compile, and run a simple random number generator program. Write these points into a file. Analyze them with a simple histogram to make sure they look reasonable.

- **Part a** Write a program to generate random numbers on the interval $[-1, 1]$
  1. You might already have code to solve this problem. If not, your code could look like this

```c
#include <stdio.h>
#include <stdlib.h> // needed to use the function atoi
#include <time.h>   // needed to use time(0)

int main(int argc, char **argv) {

  srand(time(NULL)); // initialize the random number generator

  int N = atoi( argv[1] ); // convert command-line input to N = number of points

  for (int i=0;i<N;++i) {

    // randomly sample from [0,1]
    double x = ((double) rand())/((double) RAND_MAX);
    // scale the interval to be of length 2 -- it is now [0,2]
    x = 2.*x;
    // shift the interval to be [-1,1]
    x = x - 1.;

    printf("x on [-1,1] = %lf\n",x);
  }
  return 0;
}
```

  2. Compile and run this program. Make sure it works.
- **Part b** Output points to a file
  1. Write the points x into a file (follow last weeks lecture). This will require opening a file, using fprintf within the for-loop, and closing the file.
- **Part c** Analyze the points
  1. Read the points into python, matlab, or something similar. Make a histogram using, say, 100 bins. Confirm by eye that they are roughly uniform (you might need a lot of points). If you are using Python, your code might look something like:

```python
import numpy as np
from matplotlib import pyplot as plt

x = np.loadtxt("rand_numbers.dat")
plt.hist(x,bins=100)
plt.show()
```

II. TASK #2: SURPRISING NUMERICAL LINEAR ALGEBRA (5 POINTS)

Consider the matrix equation

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}\,,$$

where

$$\mathbf{A} = \begin{pmatrix} 0.7073725 & 0.5204556 \\ 0.8158208 & 0.6002474 \end{pmatrix}$$

and

$$\mathbf{b} = \begin{pmatrix} 0.1869169 \\ 0.2155734 \end{pmatrix}.$$

The exact solution is

$$\mathbf{x} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

Consider two approximate solutions

$$\mathbf{x}_1 = \begin{pmatrix} 0.9999999 \\ -1.0000001 \end{pmatrix}, \qquad \mathbf{x}_2 = \begin{pmatrix} 0.4073666 \\ -0.1945277 \end{pmatrix}.$$

You will write code to show the worst solution, $\mathbf{x}_2$, actually does a better job at solving this $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ than $\mathbf{x}_1$. Why? Google "matrix condition number" to find out more (our matrix has a very large condition number of about 10^{14}). Note that this issue has nothing to do with floating-point numbers – its a property of ill-conditioned matrices.

- **Part a** Write a program to take the matrix-vector dot product of $\mathbf{A} \cdot \mathbf{x}$ for different values $\mathbf{x}$

  1. Use the starter code I've written below, fill in where you are asked to.

```c
#include <stdio.h>
int main() {

  // declare vectors here
  double x[2], b[2];
  x[0] = 1.0;
  x[1] = -1.0;

  double A[2][2];
  // add code for a matrix A here

  // matrix-vector dot product... "input" is A and x and "output" is b = A*x
  for(int i=0;i<2;i++){
    b[i] = 0.;
    for(int j=0;j<2;j++){
      b[i] += A[i][j]*x[j];
    }
  }

  // now print b to screen here

  return 0;
}
```

  2. Make sure to compile and run the code.
  3. Test your code: let $\mathbf{A}$ be given above and use the exact solution for $\mathbf{x}$. Then $\mathbf{A} \cdot \mathbf{x}$ should be exactly $\mathbf{b}$.

- **Part b** Add code to compute the residual $\mathbf{r} = \mathbf{b} - \mathbf{A} \cdot \mathbf{x}$ and print its value to screen. Please print using scientific notation!

- **Part c** Use your code to compute $\mathbf{r}_1$ and $\mathbf{r}_2$ corresponding to these two approximate solutions, $\mathbf{x}_1$ and $\mathbf{x}_2$. Since you will need to reuse the matrix-vector dot product code, you can either (i) copy and paste this code (easy way, should work) or (ii) make the dot product code a function (harder, but better, way).

Run your code and confirm that the residual $\mathbf{r}_1$ is larger than $\mathbf{r}_2$, despite the fact that $\mathbf{x}_2$ is closer to the true solution than $\mathbf{x}_1$.

When your code finishes running it should output both residuals to screen. The result should be alarming: a clearly bad solution almost solves the matrix equation.

III. UPLOAD YOUR WORK TO GIT

Congratulations! You've completed the lab. Now lets upload your work to git.

Follow the same series of steps as you've done for the last few labs. Place all work for this lab (task 1: random sampler code, python analysis script, and histogram; task 2: a single program) into a folder called "Lab4" and push it to bitbucket.